Parveen Khurana

124 Followers About Following

Sigmoid Neuron and Cross-Entropy

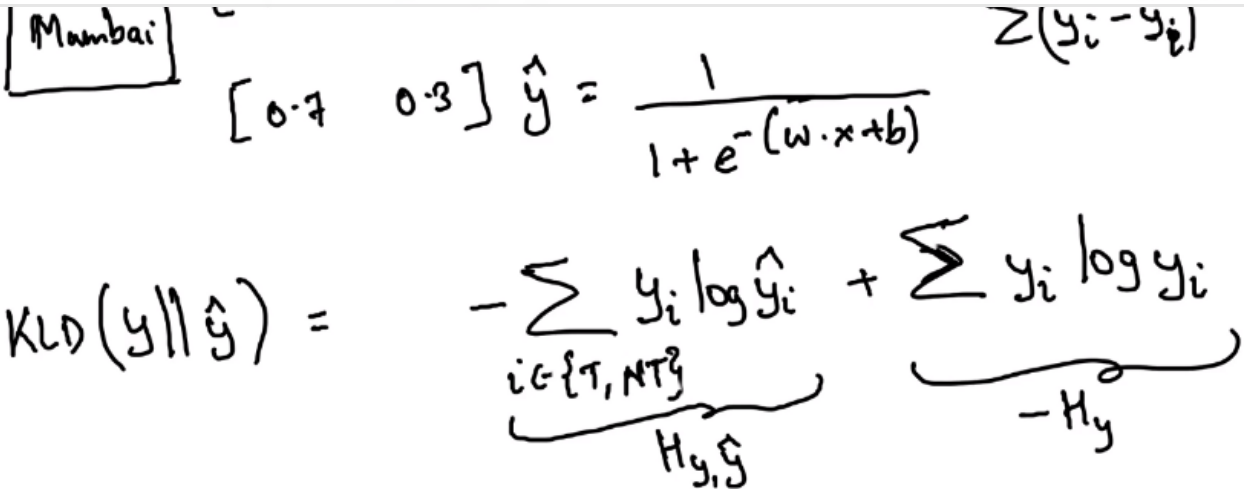
Parveen Khurana Jan 6, 2020 · 5 min read

This article covers the content discussed in the Sigmoid Neuron and Cross-Entropy module of the Deep Learning course and all the images are taken from the same module.

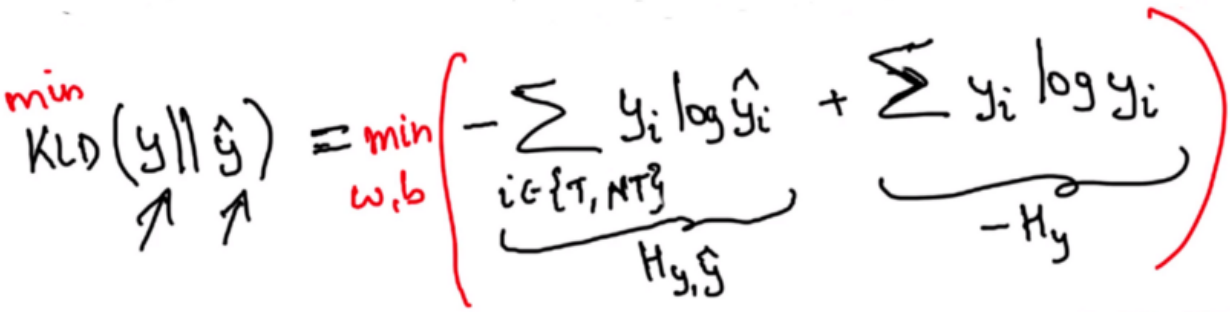
The situation that we have is that **we are given an image**, **we know the true label for that image if it contains text or not** and in the below case since the image contains text, we can say that all the probability mass is on the random variable taking on the value 1 and there is 0 probability mass on the random variable taking on the value No Text.

Of course, **in practice, we don't know this true distribution**, so **we are approximating the same using the sigmoid function**, and when we pass this input as **x** to the sigmoid neuron, we get the output to say 0.7 which we can again interpret as the probability distribution as the probability of the image containing text is 0.7 and the probability of the image not containing text is 0.3.

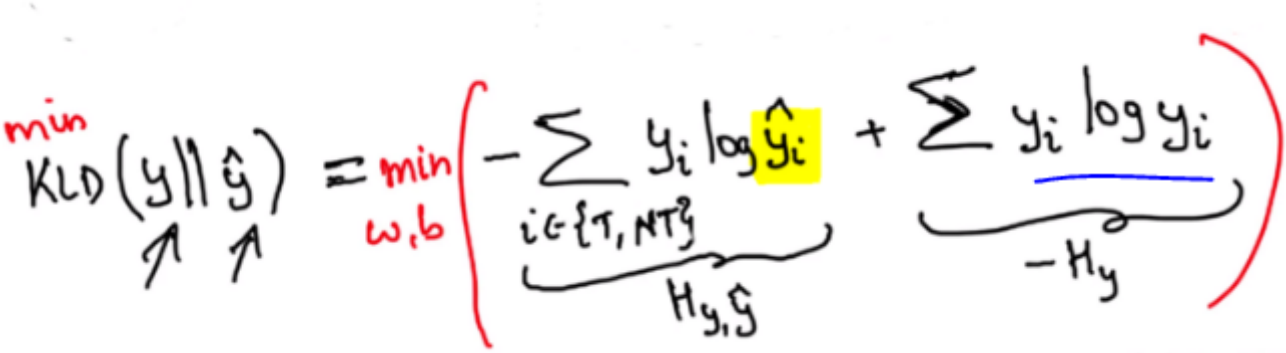
So, we were computing the difference between these two distributions using the squared error loss but now we have a better metric, something which is grounded in probability theory which is the **KL Divergence** between these two distributions.

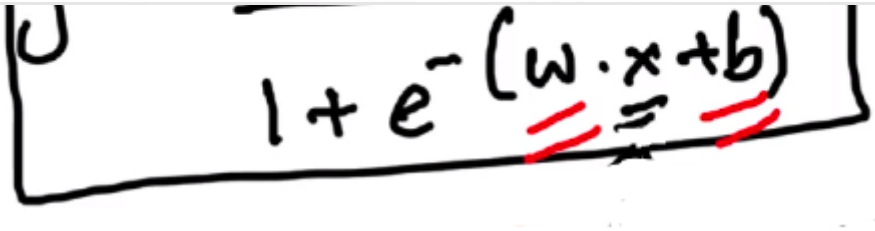


So, now instead of minimizing the squared error loss, **we are interested in minimizing the KL Divergence and this minimization would be in respect to the parameters of the model(w, b)**

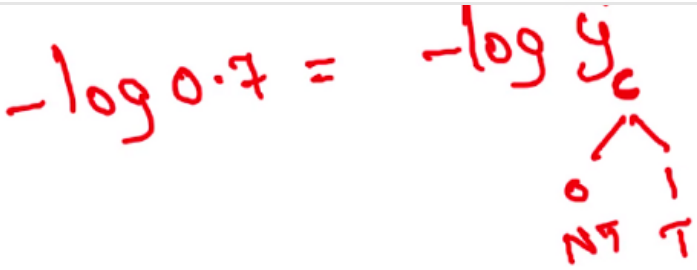


The term highlighted(in yellow) in the below image depends on w, b as per the Sigmoid definition/equation and **the term underlined in blue does not depend on w and b**





So, our goal is to minimize a quantity with respect to the parameters w, b. And since the blue underlined part in the above image does not depend on w, b; we can think of this as a constant. And **our task reduces to just minimizing the first term i.e Cross Entropy with respect to parameters w, b**.



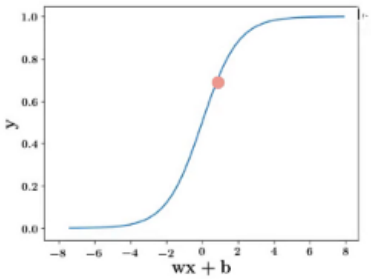
So, the **value of Cross-Entropy** in the above case turns out to be: -log(0.7) which is the same as the **-log of y_hat for the true class**. (True class, in this case, was 1 i.e image contains text, and **y_hat** corresponding to this true class is 0.7).

Using Cross-Entropy with Sigmoid Neuron

$x = image$

$y = [0, 1]$

$$\hat{y} = \frac{1}{1 + e^{-(w \cdot x + b)}}$$



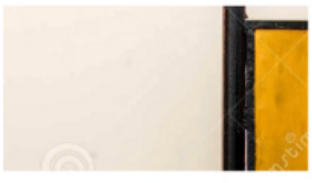
$\hat{y} = 0.7$

$\tilde{y} = [0.3, 0.7]$

$$\mathscr{L}(\theta) = -\Sigma_{i \in \{0,1\}} y_i \log \tilde{y}_i$$

$$\mathscr{L}(\theta) = -(y_0 \log \tilde{y}_0 + y_1 \log \tilde{y}_1)$$

$$\mathscr{L}(\theta) = -(y_0 \log (1 - \tilde{y}_1) + y_1 \log \tilde{y}_1)$$



$$\tilde{y} = [0.3, 0.7] \qquad \tilde{y} = [0.8, 0.2]$$

$$\mathscr{L}(\theta) = -(y_0 \log(1 - \tilde{y_1}) + y_1 \log \tilde{y_1})$$

$$\mathscr{L}(\theta) = -(0 * \log 0.3 + 1 * \log 0.7) = -\log 0.7 = -log\ \hat{y}$$

$$\mathscr{L}(\theta) = -(1 * \log 0.8 + 0 * \log 0.2) = -\log 0.8 = -log(1 - \hat{y})$$

When the true output is 1, then the Loss function boils down to the below:

$$-log\ \hat{y}$$

And when the true output is 0, the loss function is:

$$-log(1 - \hat{y})$$

And this is simply because there is 1 term which gets multiplied with 0 and that term would be zero obviously, so what remains is the loss term.

A more simplified way of writing Cross-Entropy Loss function:

$$y = 1 \qquad y = 0$$

$$\hat{y} = 0.7 \qquad \hat{y} = 0.2$$

$$\mathscr{L}(\theta) = -\log \hat{y} \qquad if\ y = 1$$

$$\mathscr{L}(\theta) = -((1-y)\log(1-\hat{y}) + y\log\hat{y})$$

Learning Algorithm for Cross-Entropy Loss function

I/P	O/P
1.2	0
-2.1	1
3.2	0
-0.5	1
0.6	1

$$\hat{y} = \frac{1}{1+e^{-(wx+b)}}$$

As is clear from the output, we are dealing with the Classification problem(as the possible output is 0, 1). The use of Cross-Entropy Loss only makes sense in the Classification case because that's when we are trying to interpret the output as a probability. We compute the output for each of the 5 data points and use it to compute the loss function as:

$$\mathscr{L}(\theta) = -\sum_{i=1}^{5}((1-y_i)\log(1-\hat{y}_i) + y_i\log\hat{y}_i)$$

And once we have the Loss, we can compute the delta terms(highlighted in below image) and update the parameters and continue this over multiple iterations over the

Initialise w, b

Iterate over data:

$compute\ \hat{y}$

$compute\ \mathscr{L}(w, b)$

$w_{t+1} = w_t - \eta \Delta w_t$

$b_{t+1} = b_t - \eta \Delta b_t$

till satisfied

And we compute these δ(delta) terms by taking the partial derivatives of the loss function with respect to w and b.

Computing partial derivatives with cross-entropy loss:

The loss function(in general) we can write as:

$$\mathscr{L}(\theta) = -[(1-y)\log(1-\hat{y}) + y\log\hat{y})]$$

And then we can compute the partial derivative of the loss function with respect to w using chain rule as:

$$\Delta w = \frac{\partial \mathscr{L}(\theta)}{\partial w} = \frac{\partial \mathscr{L}(\theta)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w}$$

$$\frac{\partial \mathscr{L}(\theta)}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}}\{-(1-y)\log(1-\hat{y}) - y\log\hat{y})\}$$

$$= (-)(-1)\frac{(1-y)}{(1-\hat{y})} - \frac{y}{\hat{y}}$$

$$= \frac{\hat{y}(1-\hat{y}) - y(1-\hat{y})}{(1-\hat{y})\hat{y}}$$

$$= \frac{\hat{y}-y}{(1-\hat{y})\hat{y}}$$

$$\frac{\partial \hat{y}}{\partial w} = \frac{\partial}{\partial w}\left(\frac{1}{1+e^{-(wx+b)}}\right)$$

$$= \frac{-1}{(1+e^{-(wx+b)})^2}\frac{\partial}{\partial w}(e^{-(wx+b)}))$$

$$= \frac{-1}{(1+e^{-(wx+b)})^2} * (e^{-(wx+b)})\frac{\partial}{\partial w}(-(wx+b)))$$

$$= \frac{-1}{(1+e^{-(wx+b)})} * \frac{e^{-(wx+b)}}{(1+e^{-(wx+b)})} * (-x)$$

$$= \frac{1}{(1+e^{-(wx+b)})} * \frac{e^{-(wx+b)}}{(1+e^{-(wx+b)})} * (x)$$

$$= \hat{y} * (1-\hat{y}) * x$$

$$\Delta w = (\hat{y} - y) * x$$

And similarly, the partial derivative of the loss function with respect to b would be:

$$(\hat{y} - y)$$

Changes in the Code for Cross-Entropy Loss function:

Our code with the squared error loss function is as below:

```
X = [0.5, 2.5]
Y = [0.2, 0.9]
```

```
def error(w, b):
     err = 0.0
     for x, y in zip(X, Y):
        fx = f(w, b, x)
        err += 0.5* (fx - y) ** 2
     return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta = -2, -2, 1.0
    max_epochs = 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Now three functions would change: first one is error function and the other two are the functions that compute the derivative of the loss function with respect to w and b.

In the error function, earlier we were computing the squared error loss which is now changed to cross-entropy loss:

Error function:

Earlier

```
def error(w, b):
     err = 0.0
     for x, y in zip(X, Y):
        fx = f(w, b, x)
        err += 0.5* (fx - y) ** 2
     return err
```

Updated one

```
def error(w, b):
     err = 0.0
     for x, y in zip(X, Y):
        fx = f(w, b, x)
        err += -[(1 - y) * math.log (1 - fx , 2) + y * math.log (fx, 2)]
```

Other two functions:

Earlier

```
def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x
```

Updated one

```
def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y)
```

```
def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * x
```

Machine Learning Deep Learning Artificial Intelligence Data Analytics Cross Entropy